IN THE SPECIFICATION:

Please replace the paragraph beginning at page 4, line 8 with:

According to one embodiment of the invention, two different cyclical redundancy check (CRC) values are obtained for each record key to be indexed. The first CRC value defines the portion of the index table in which the record's address is stored. The second CRC value is then stored with the record's address in the table. To then retrieve a record based upon a search key, the key's first CRC value narrows the portion of the index table to be searched. Once a specific portion of the index table is identified with the first CRC value (which can be thought of as a table within a table), that portion of the table can be sequentially searched until the second CRC value for the search key is located. The second CRC value then identifies the memory address of the record containing a matching record key. Because CRC values are collision resistant, the chance of collision within an index table using at least two types of CRC values is very low. Further, because CRC values can be quickly calculated using tables, the processing time for calculating an index value is reduced. Also, in addition to using CRC values obtained from a record's key to index that record's address, various embodiments of the invention also employ CRC values from a record's key to position that record among a plurality of different storage media.

Please replace the 4 paragraphs beginning at page 7, line 12 with:

More particularly, the address translation of the CRC-CCITT value of the key provides a "start" location in the index table. A linear search for an indexed CRC-16 value matching the CRC-16 value of the key then begins from this start location. This linear search is efficient because the data to be searched is sequential and minimal in size. The four-byte offset for each record in the database is stored with the CRC-16 value of its key. Thus, when an indexed CRC-16 value matching the CRC-16 value of the key is found, the associated four-byte offset is used to retrieve the record. The search key is then compared with the record's key, to confirm that they are the same. ~~If the keys do not match, a collision has occurred (because two keys in the database have the same hash value) and the search of the index table continues until the correct record is retrieved. Due to the uniqueness of the four-byte hash value, however, collisions rarely occur.~~

A database index table 101 that implements this hybrid method is shown in Fig. 1. The database indexed by this particular embodiment of this invention uses variable width database records to reduce the amount of memory space required, but fixed width records may alternately be employed as will be discussed below. The index table is initially formed by a plurality of fixed-size index table clusters 103_0, 103_1, ... 103_N, where illustratively in Fig. 1 N equals 66635. Using fixed-sized clusters permits easier record addition and deletion, but variable-sized clusters may alternately be used to reduce the amount of required memory space or to reduce overflow problems.

Stated in other words, t~~T~~he particular embodiment shown in Fig. 1 with 2^{16} (i.e., 65,536) initial data clusters 1030 to 10365535 (i.e., data clusters created to initially form the index table). Thus, the index conveniently has one initial data cluster 103_i corresponding to each possible CRC-CCIT value. Other embodiments, however, may employ fewer or greater numbers of initial data clusters ~~103, however~~. Further, as will be explained in detail below, ~~additional~~ overflow data clusters ~~103~~ may be added to the index table.

Each index table cluster 103_i ~~shown in Fig. 1~~ contains an array of ~~four~~ entries. Fig. 1 illustrates clusters with four entries in which records, or tuples, can be stored, as illustrated by tuples, 105a - 105d of cluster 103_1, but the number of entries ~~K~~ may be varied to optimize the performance of the index table 101, as will be explained in detail below. Each entry ~~105~~ in the index table has one two-byte CRC-16 field (e.g., field 107 of cluster 103_1) and one four-byte record offset field (e.g., field 109 of cluster 103_1). In this particular embodiment, unused entries have the CRC-16 field ~~107~~ set to 0 and the offset field 109 set to its maximum value (MAXVALUE), but other values can alternately be employed.

While the entries ~~105a-105c~~ of each index table cluster are available to store CRC values and record offsets, the last entry ~~105d~~ of each index table cluster ~~103~~ is reserved for use as a link pointer. When all but the last entry ~~105d~~ in an index table cluster ~~103~~ ~~is~~ are filled, and a new record offset value needs to be added to the index table cluster, an overflow index table cluster ~~103~~ is created. The address of this overflow index table cluster 103 is then stored in the last entry ~~105d~~, and the new record offset value is stored in the first entry ~~115a~~ of the overflow index table cluster. For example, as shown in Fig. 1, when entries 105a-105c

of initial index table cluster 103_1, are filled, the overflow index table cluster 103_{1A} is created, and the starting address of this overflow index table cluster 103_{1A} is stored in entry 105d of initial index table cluster 103_1. As will be understood by those of ordinary skill in the art, if all but the last entry ~~entries 105a-105c~~ of the overflow index table cluster 103_{1A} are filled, a second overflow index table cluster 103_{1B} (not shown) is created, and the starting address of this overflow index table cluster 103_{1B} is stored in the last entry ~~105d~~ of ~~initial~~ index table cluster 103_{1A}. As will also be appreciated by those of ordinary skill in the art, additional overflow tables can be created and linked to full overflow tables as necessary. Thus, additional CRC-16 values and their corresponding record offsets that cannot be stored in a full initial index table cluster ~~103~~ can be stored in an overflow index table cluster ~~103~~ directly or indirectly linked to the ~~first~~ full index table cluster ~~103~~.